

Featherweight Java

*Proprietà di Type Safety:
dimostrazione*

Un problema sui cast (continua)

Per dimostrare la *type preservation* con una *small-step semantics* è necessario (solo formalmente), introdurre gli *stupid-cast*

Ad esempio, siano A e B sottoclassi di OBJECT, ma non correlate fra loro: si consideri il seguente termine

(A) ((Object) new B())

che è ben tipato per il compilatore.

Ma run-time può succedere che ...??????????????

- **(Object) new B() → (A) new B()**

chiaramente **(A) new B()** non è tipabile con nessuna delle
2 regole descritte sopra

Regola per stupid-cast

Per trattare formalmente il teorema di Type-Preservation (senza passare alla big-step semantics) , estendiamo le regole di Typing con la seguente regola **(T-SCast)**

$$\Gamma \mid \text{---} t : D \quad \text{not } (C <: D) \quad \text{not } (C <: D)$$

stupid warning

$$\Gamma \mid \text{---} (C) t : C$$

Type Preservation

Vogliamo dimostrare che la well-typedness è preservata durante la computazione.

Nel nostro caso, dimostreremo, in particolare, che se un termine ha un tipo allora il termine ottenuto con un passo di riduzione è tipabile con un tipo più piccolo.

Per condurre la prova in modo formale, abbiamo bisogno di alcuni Lemmi Preliminari.

Inoltre useremo sempre implicitamente la proprietà di *weakening*:

se $\Gamma \vdash t : D$ allora $\Gamma, x:C \vdash t : D$

(ossia posso sempre considerare un contesto con altre ipotesi aggiuntive che non sono usate nella derivazione di tipo)

Lemmi preliminari per la type safety

Lemmi.

(1) Se $\text{mtype}(m; D) = C' \rightarrow C''$ e $C <: D$
allora $\text{mtype}(m; C) = C' \rightarrow C''$

(2) Se $\text{mtype}(m; D) = C' \rightarrow C''$ e $\text{mbody}(m; D) = (x, t)$
allora per qualche D' e qualche C , con $D <: D'$ e $C <: C''$,
abbiamo che

$$x:C', \text{this}:D' \mid\!\!\!\!-\! t : C$$

La prova formale è per induzione su $C <: D$ per
Lemma 1 e sulla derivazione di $\text{mbody}(m; D)$ per Lemma 2.
Esercizio per gli studenti: prova formale.

Substitution Property

Molto più interessante è la prova della proprietà di Sostituzione: *se in una derivazione di tipo rimpiazziamo a variabili termini, tali che questi termini hanno tipi più piccoli di quelli assunti per le variabili, allora otteniamo ancora una deduzione valida di tipo che assegna al nuovo termine finale un tipo più piccolo.*

Lemma (Substitution Property)

Se $\Gamma, x:B \mid \text{---} t : D$ e $\Gamma \mid \text{---} s : A$ con $A <: B$, allora

$\Gamma \mid \text{---} t' : C$ con $C <: D$,

dove t' è il termine ottenuto da t rimpiazzando x con s .

Prova: per induzione sulla derivazione di $\Gamma, x:B \mid \text{---} t : D$.
(Esercizio alla lavagna)

Proprietà: Type Preservation

(*Preservation*)

Se $\Gamma \mid\!\!\!-\ t : C$ e $t \rightarrow t'$ con una regola di valutazione,

allora

$\Gamma \mid\!\!\!-\ t' : C'$ per qualche $C' < C$.

Prova. Per induzione sulla derivazione di $t \rightarrow t'$ (Dim. alla lavagna, sul testo).

Type Preservation

Typechecking: Un termine con un sottotipo può essere usato in ogni contesto in cui è richiesto un tipo più grande (polimorfismo) →

I termini (programmi) ben tipati si mantengono ben tipati durante la valutazione.

Nota sui Cast

Un termine FJ ben tipato che non contiene down-cast si riduce sempre a termini che non contengono nè down-cast nè *stupid* cast.

Proprietà: Progress

- **GOAL** (*Progress*)

Un termine chiuso ben tipato, che non contiene down-cast, o è un valore o si può ridurre con una regola di valutazione

(ossia, un termine che ha un tipo e contiene solo up-cast non può produrre fallimento: o è già un valore o si riduce per valutazione)

Progress (prova formale)

Lemma (assenza di “campo inesistente” o di “messaggio non compreso” in programmi ben tipati)

Sia t un termine ben tipato:

- Se $t = \text{new } C(t_1, \dots, t_n).f$ allora
 f appartiene ai $\text{fields}(C)$
- Se $t = \text{new } C(t_1, \dots, t_n).m(s_1, \dots, s_k)$ allora
 $\text{mbody}(m, C) = ((x_1, \dots, x_k), t_0)$

Prova. Triviale, per induzione sulle regole di Typing

Progress (prova formale)

Si deve formalizzare il concetto di “contesto di valutazione” per esprimere “il prossimo termine da ridurre”.

Un contesto di valutazione E è un buco $[]$ oppure $E.f$, $E.m(t)$, $v.m(v,E,t)$, $\text{new } C(v,E,t)$, $C(E)$.

Scriviamo $E[t]$ per indicare il termine ottenuto rimpiazzando il buco in E con t .

Così, se $t \rightarrow t'$, possiamo scrivere

t come $E[r]$ e t' come $E[r']$, per un unico E ,

dove $r \rightarrow r'$ con un passo di valutazione

Progress : teorema finale

Formulazione formale del Teorema di
Progress:

Sia t un termine chiuso che non può essere ridotto(forma normale); allora o t è un valore oppure t è uguale a un contesto di valutazione E riempito con $(C)\text{new } D(v)$ dove D non è sottotipo di C ossia

$t = E[(C)\text{new } D(v)]$ con D non sottotipo di C .

Prova. Per banale induzione sulle regole di typing (Esercizio alla lavagna)

Type Safety : tiriamo le fila

Cucendo insieme i risultati ottenuti:

- **Se un termine FJ è ben tipato ed eseguiamo un passo di valutazione, otteniamo un termine ben tipato (*Preservation*)**
- **Se un termine è ben tipato, allora ogni selezione di campo ed invocazione di metodo è corretta (non può produrre uno stuck) (*Lemma*)**
- **Se arriviamo ad un termine stuck (non si può ridurre e non è un valore) allora ci siamo fermati per un Cast in cui il tipo effettivo dell'oggetto non è un sottotipo del target del Cast (*Progress formale*)**

Come volevasi dimostrare : Type Safety

TYPE SAFETY

**Un programma corretto rispetto ai tipi
senza down-cast non può fallire run-
time**

Prova. Preservation + Progress .

*Ovviamente in Java sono ammessi down-
cast, in questo caso l'eventuale
fallimento è rappresentato dal
sollevamento di opportune eccezioni
run-time.*

Esempio

Siano A e B due classi in cui il metodo m è riscritto, es.:

```
class A extends Object { A() { super(); }  
                        Bool m() { return true; } }  
class B extends A      { B() { super(); }  
                        Bool m() { return false; } }
```

Si consideri il termine ((A) new B()).m

- *Typing:* $\vdash (A) \text{ new } B() : A$ *ossia*
 $\vdash ((A) \text{ new } B()).m : \text{Bool}$

(2) Valutazione: $(A) \text{ new } B() \rightarrow \text{new } B()$ *perciò*

$((A) \text{ new } B()).m \rightarrow \text{new } B().m$

che restituisce false (binding dinamico)

(3) Type Preservation : $(A) \text{ new } B() \rightarrow \text{new } B()$

$\vdash (A) \text{ new } B() : A$, $\vdash \text{new } B() : B$ *dove* $B <: A$

Una digressione: *Interfacce & Condizionali*

In FJ mancano sia le *Interface* che i *Condizionali*: aggiungerli ambedue pone qualche problema nella formulazione di un typing algoritmico

Condizionali Java

$$\Gamma \vdash t_1:\text{boolean} \quad \Gamma \vdash t_2:T_2 \quad \Gamma \vdash t_3:T_3$$

$$\Gamma \vdash t_1 ? t_2 : t_3 : \quad ?$$

La regola semplice che richiede che i tipi dei due rami siano uguali diventa problematica in presenza del subtyping, per cui ogni oggetto di un tipo ha anche ogni suo sopratipo: i due tipi T_2 e T_3

potrebbero avere diversi sopratipi in comune....

come determinare in tal caso l'unico tipo del condizionale?

Subtyping e join

Versione algoritmica del subtyping

$$\frac{\Gamma \vdash t_1:\text{boolean} \quad \Gamma \vdash t_2:T_2 \quad \Gamma \vdash t_3:T_3}{\Gamma \vdash t_1 ? t_2 : t_3 : \text{join}(T_2, T_3)}$$

dove $\text{join}(T_2, T_3)$ è il minimo fra i supertipi comuni di T_2 e T_3

Ma questo richiede l'esistenza dei join, ossia se due tipi hanno supertipi in comune hanno anche un minimo fra questi!

Aggiungiamo le Interfacce

Aggiungere le Interface è semplice(*esercizio alla lavagna*)
: ma questo introduce una sorta di ereditarietà multipla:

Es. **interface A.....**
 interface B.....
 interface C extends A,B
 interface D extends A,B.....

Allora A e B hanno i sopratipi C e D ma non un minimo (C e D sono inconfrontabili)

Regola Java per i Condizionali

$$\Gamma \vdash t_1:\text{boolean} \quad \Gamma \vdash t_2:T_2 \quad \Gamma \vdash t_3:T_3 \quad T_2 \leq T_3$$

$$\Gamma \vdash t_1 ? t_2 : t_3 : T_3$$
$$\Gamma \vdash t_1:\text{boolean} \quad \Gamma \vdash t_2:T_2 \quad \Gamma \vdash t_3:T_3 \quad T_3 \leq T_2$$

$$\Gamma \vdash t_1 ? t_2 : t_3 : T_2$$

Sembra molto ragionevole....ma pone dei problemi nella type preservation con la semantica small-step!

Esercizio: trovare controesempio